REMARKS

Claim 5 is amended to correct an informality and is not amended in response to any rejection.

Claims 1-4, 11-14 and 21-24 are rejected under 35 USC 103(a) as being unpatentable over U.S. Patent 5,297,150 (CLARK) in view of U.S. Patent 5,954,925 (SHIMOMURA). The Examiner is respectfully requested to withdraw the rejections of these claims in view of the following comments distinguishing the rejected claims over CLARK and SHIMOMURA.

Claim 1

The applicant's invention relates to a method for helping a programmer debug program code after the program code has been executed and the programmer has found that at some point during program execution one of the variables the program controls has taken on an unexpected value. This happens when one or more of statements (lines of program code) are incorrect. The statement that directly sets the value of that variable (the "error variable") to an incorrect value itself may or may not be the incorrect statement. For example the incorrect statement might be a line of program code that sets the value of another variable upon which the error variable depends. The invention as recited in claim 1, receives an indication of the error variable and then determines the "error set" of the error variable. The error set is the set of all statements in the program code that influence (i.e., that "are relationally connected to") the value of the error variable. The invention then assigns a priority value to each statement in the error set indicating the probability that the statement is the source of the error in the value of the error value. The invention then displays each statement in the error set in a manner indicating its relative priority value. This helps the user to more quickly locate the incorrect statement(s) because it indicates an order in which the user should inspect the statements for errors.

CLARK discloses a method for determining which "flow paths" in a program are the most likely to fail. A "flow path" is not a single line of code but is instead a sequence of lines of code that a program may execute between the programs starting and ending statements. There can be many possible flow paths because most programs contain conditional statements such as "IF" statements which can direct program flow to any of several statements depending, for example, on the value of some variable at the time the conditional statement is executed. CLARK teaches to create a graph (such as shown in FIG. 2) of a program to help identify program flows. The numbered circles represent program statements and the arrows represent possible directions of program flow. Statements 1 and 3 are the starting and ending statements of the program. Statements such as statement 2 from which two or more arrows extend are conditional statements (such as "IF" statements) that can direct program flow to any of two or more statements depending on the value of some variable at the time the conditional statement is executed. From the graph of FIG. 2 it is easy to see that the program may follow any of several flow paths from starting statement 1 to ending statement 3. For example one flow path is 1-2-3 while another flow path is 1-2-12-8-9-10-11-2-3.

According to CLARK the flow paths are more likely to cause an error during program execution and are more difficult to debug when they include a large number of relatively complex statements. CLARK therefore teaches a method for analyzing a program, identifying each possible flow path, and then ranking the identified flow paths according to their complexity.

The applicant's claim 1 recites a first step utilizing an input system to indicate an "error variable containing an error value that differs from a first desired value." The applicant's specification paragraph 0008 indicates an error variable is any variable in the program that takes on a value that differs from a desired value. Thus, the applicant's method of claim 1 requires the identification of an "error variable" that took on an undesirable value during program execution. Since the object of the applicant's invention is to help a programmer determine which line of program code caused the error, the applicant's method needs to know which program variable had the wrong value.

CLARK does not teach a method including a step of receiving an indication of an error variable because CLARK's teachings have nothing to do with helping a programmer determine which statement actually caused an error in the value of a particular program variable during program execution. CLARK's teaching relates only to helping a programmer determine which flow paths in a program are the most complex and therefore more likely to cause an error when the program is executed. Hence it isn't necessary to execute a program before applying CLARK's method and it isn't necessary to CLARK's method to be aware of the identity of an error variable having an undesired value during program execution.

Thus CLARK fails to teach the first step of the applicant's claim 1, "utilizing the input system to indicate an error value in the program code containing an error value that differs from a first desired value". The Examiner does not cite CLARK as teaching this step but does cite SHIMOMURA col. 8, lines 5-12 as teaching this step and suggests that it would be obvious to combine this teaching of SHIMOMURA with CLARK. The cited section of SHIMOMURA talks about a system that displays program input and output variable values while executing a program. When a user clicks on a displayed variable that is in error, the system records the variable name, its value and the point during the program at which the error occurred. The Examiner indicates it would be obvious to use SHIMOMURA system to provide an error variable name as input to CLARK's system as recited in claim 1. However as discussed above, CLARK's system does not process error variables because it is not concerned with determining which lines of code caused a variable to take on an undesirable value during program execution. It is only concerned with determining which paths through a program are the most complex. Hence it would not be obvious to use SHIMOMURA's system to provide CLARK's system with input information for which CLARK's system has no use.

The second step of the applicant's claim 1 is to "obtain an error set of the error variable … comprising a subset of the program code statements … [that are] relationally connected to the error variable". These are the program code statements that can influence the value or the error variable. As noted above, CLARK's system is unconcerned with an error variable, and therefore cannot and does not determine a set of code statements that are relationally connected to an error value as recited in claim 1. The Examiner cites CLARK col. 4 lines 14-16 as teaching obtaining an error set, but that section of CLARK does not do that. Note that an error set is the set of program statements that are relationally connected to (can influence the value of) an error variable. CLARK's system does not care about which program statements can influence the value of a variable. CLARK's system is concerned with identifying sets of program statements that form various "flow paths" and does not select statements in a flow path based on whether the statements affect the value of some variable that was found to be in error during program execution. The cited section of CLARK teaches only that if the program is to be tested, CLARK's system can determine which program flows are most complicated and therefore most likely to contain a bug. In the context of the rest of CLARK's patent, it is clear that CLARK means the program flows most likely to contain a bug are the program flow that are most complex, not the program flows containing statements that are relationally connected to some program variable found to have an incorrect value during program execution. Thus CLARK fails to teach the second step ("obtaining an error set …") of the applicant's claim 1.

The applicant's claim 1 further recites a third step of "assigning a priority code to each statement in the error set indicating a computed probability that the associated program code statement is an error source of the error variable". The Examiner cites CLARK col. 3, lines 34-40 as disclosing this step, however as discussed above, CLARK does not teach establishing an error set (a set of statements relationally connected to an error variable) and therefore does not teach assigning a priority code to statements in an error set. The cited section of CLARK teaches assigning weights to nodes (i.e., program statements) based on their complexity, not on a probability that they are the source of an error in a value of a variable that occurred during program execution. Thus CLARK fails to teach the recited third step of claim 1.

Claim 1 further recites a fourth step of "[presenting] each program code statement in the error set in a manner that indicates the relative ordering of the priority value of the program code statement…" The Examiner cites CLARK col. 6, lines 33-35 as disclosing this step, but this section of CLARK talks about displaying program flows, not program statements, in order of priority. Recall that a program flow is a sequence of statements that a program might execute, and that CLARK prioritizes them in accordance with their complexity. The fourth step of claim 1 relates to program statements (not program flows) that are prioritized in order in which they are likely to have caused an error in a particular variable during a program execution. Thus CLARK fails to teach the fourth step of the applicant's claim 1.

Claim 1 is therefore patentable over the combination of CLARK and SHIMOMURA since the cited references fail to disclose or suggest any steps 2-4 of claim 1 and because it would not be obvious to combine the teachings of CLARK and SHIMOMURA.

Claim 2

Claim 2 is patentable over the cited references for reasons similar to those discussed above in connection with claim 1. Claim 2 further recites presenting program statements in the error set in an order from a most likely error source to a least likely error source. The Examiner cites CLARK FIG. 7 and col. 6, lines 38-40 as teaching this step. However the cited section of CLARK teaches displaying a solution set that ranks flow paths in order of decreasing complexity. CLARK does not teach displaying a set of program statements in an order in which they are most likely to be the source of an error in the value of a particular program variable.

Claim 3

Claim 3 depends on claim 1 and is patentable over the cited references for reasons discussed above in connection with claim 1. CLAIM 1 further recites, "each program statement in the error set is presented with the associated priority value." The Examiner cites CLARK, FIG. 7 and column 6, lines 38-40 as teaching this. FIG. 7 of CLARK shows a display of a graph of program statements with each statement being presented with an associated weight. CLARK's program statement "weight", however, has nothing to do with a program statement's "priority value" as defined by the applicant's claim 1. Claim 1 defines the priority value of a program statement as its "computed probability that [the program statement] is an error source of the error value", i.e., the source of an error in the value of a particular variable occurring during program execution. As taught in CLARK's col. 3, lines 34-36, CLARK's "weights" quantify the complexity of each statement, not the probability that the statement is the source of an error in a particular variable as recited in claim 3. Thus claim 3 is patentable over the cited references.

Claim 4

Claim 4 depends on claim 1 and is patentable over the cited references for similar reasons. Claim 4 further recites the step of generating the error set, "every program code statement in the error set being in the execution set and also being relationally connected to the error variable." Thus the error set includes any program code statement having an influence on the value of a variable that was in error during program execution. The Examiner cites CLARK, col. 6, lines 22-32 as teaching this. However these lines of CLARK talk only about identifying flow paths and have nothing to do with generating an "error set" as recited in claim 4.

Claim 4 further recites, "obtaining an execution set comprising all program code statements that are executed in the error cycle." Claim 4 defines an error cycle as a program execution cycle in which the error variable obtains an error value. Thus Claim 4 recites determining a set of all program code statements that were executed during some execution cycle of a program in which a variable took on an undesirable value. The Examiner correctly indicates that CLARK does not teach this although cites CLARK col. 6, lines 10-15 as being relevant to this step. However this section of CLARK teaches to search for program segment paths that are most likely to fail based on weights (based on statement complexity) of program statements in the paths and has nothing to do with determining which program code statements were executed during some program cycle in which an error occurred. Searching for complex program flows has nothing to do with searching for a set of statements that were executed during an execution cycle of a program in which an error in a variable value actually occurred.

Claim 4 recites "obtaining an error cycle in which the error variable obtains the error value" and "obtaining an execution set comprising all program statements that are executed in the error cycle". The Examiner cites SHIMOMURA FIG. 1, blocks 11 and 13, FIG. 2, col. 8, lines 13-28 and 44-47 as teaching this. SHIMOMURA (col. 8, lines 13-47) teaches to help a programmer locate a bug (error) in program code by making a program run up to a point at which an error in a variable is detected and then displaying the sequence of statements that were executed up to that point. The Examiner states it would be obvious to combine this teaching of SHIMOMURA with CLARK.

However CLARK does not teach a method that has anything to do with processing an execution set for any reason. CLARK is concerned only with identifying program flows and determining how complex they are. Since the cited section of SHIMOMURA teaches to generate data that is irrelevant to CLARK's method, it would not be obvious to combine this teaching of SHIMOMURA with CLARK.

The Examiner indicates that since CLARK col. 6, line 10, talks about "prior experience" one would be motivated to modify CLARK to use SHIMOMURA's technique for locating a program bug. However the "prior experience" CLARK talks about at col. 6, line 10 has nothing to do with locating statements that are the source of an identified program. CLARK's "prior experience" has to do with information gained by previously searching flow paths to ascertain complexity weights of statements in the flow path. See CLARK col. 6, lines 10-11 and 18-20. The Examiner further points to CLARK col. 3, lines 17 -20 as motivating one to modify CLARK's method to use error debugging as taught by SHIMOMURA. However the cited section of CLARK indicates only that an object of CLARK's invention is to identify a set of most complex program paths. It should be understood that identifying program paths that are susceptible to error (simply because they include many complex program statements) and debugging (identifying a particular statement that actually did cause an error during program) are two rather different objectives.

Claim 4 is therefore patentable over the combination of CLARK and SHIMOMURA.

Claim 11

The Examiner rejects claim 11 over CLARK and SHIMOMURA for the reasons set forth in connection with the rejection of claim 1. The applicant's remarks above distinguishing claim 4 over CLARK and SHIMOMURA also serve to distinguish claim 11 over CLARK and SHIMOMURA.

Claim 12

Claim 12 depends on claim 11 and is patentable over CLARK and SHIMOMURA for similar reasons. Claim 12 further recites "assigning a priority value to each statement in the error set, each priority value indicating a computed probability that the associated program code statement … is an error source of the error variable." The Examiner cites CLARK, col. 3, lines 34-40 as teaching this. However the cited section of CLARK teaches assigning weights to nodes (i.e., program statements) based on their complexity, not on a probability that they are the source of an error in a value of a variable that occurred during program execution.

Claim 12 further recites "each program statement in the error set present via the output system [to indicate] the relative ordering or the priority value of the program code statement." The Examiner cites CLARK col. 6, lines 33-35 as disclosing this step, but this section of CLARK talks about displaying program flows, not program statements, in order of priority. Recall that a program flow is a sequence of statements that a program might execute, and that CLARK prioritizes them in accordance with their complexity. The fourth step of claim 1 relates to program statements (not program flows) that are prioritized in order in which they are likely to have caused an error in a particular variable during a program execution.

Thus CLARK and SHIMOMURA fail to teach the additional limitations of claim 12.

Claim 13

Claim 13 depends on claim 12 and is patentable over the combination of CLARK and SHIMOMURA for similar reasons and for reasons set forth above in connection with claim 2.

Claim 14

Claim 14 depends on claim 12 and is patentable over the combination of CLARK and SHIMOMURA for similar reasons and for reasons set forth above in connection with claim 3.

Claim 21

Claim 21 recites a computer system implementing a prioritizing system carrying out steps similar to those recited in claim 11 and is patentable over the combination of CLARK and SHIMOMURA for similar reasons.

Claim 22

Claim 22 depends on claim 21 and is therefore patentable over the combination of CLARK and SHIMOMURA for similar reasons, and also for reasons set for above in connection with claim 2.

Claim 23

Claim 23 depends on claim 21 and is therefore patentable over the combination of CLARK and SHIMOMURA for similar reasons and also for reasons set for above in connection with claim 3.

Claim 24

Claim 24 depends on claim 21 and is therefore patentable over the combination of CLARK and SHIMOMURA for similar reasons and also for reasons set for above in connection with claim 4.

Claims 5-10, 15-20, and 25-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over CLARK and SHIMOMURA in further view of US patent 5,463,768 (CUDDIHY).

Claim 5

Claim 5 (as amended) depends on claim 4 and is therefore patentable over the combination of CLARK and SHIMOMURA for similar reasons.

Claim 5 further recites the program code includes a "correct variable having a value that agrees with a second desired value in the error cycle" and "obtaining a first sensitized set for the correct variable … comprising at least a program code statement … relationally connected to the correct variable, that is excluded in the first execution cycle and that is in the error set". In other words, claim 5 recites determining a set of program statements appearing in the error set that previously produced correct values for a variable. Since these statements are less likely to cause the error in the error variable, claim 5 further recites a step of "applying a scaling function to the priority value associated with the identical program code statement in the error set … setting a reduced computed probability that the associated program code statement is the error source of the error variable. The Examiner correctly points out that

CLARK and SHIMOMURA fail to teach this but incorrectly cites CUDDIHY as teaching this.

CUDDIHY relates to a method for analyzing "historical error logs" of the type shown in CUDDIHY's FIG. 2 generated by equipment which include an error code, time of error and various comments. CUDDIHY's abstract indicates that the error logs are analyzed to find common sections of an error log that are similar, labelling common sections as "blocks", weighing each block according to its value in diagnosing an equipment fault, and then using the block and weighting information in the future to help diagnose equipment faults.

The Examiner cites CUDDIHY Table 1 as teaching the "correct variables" recited in the applicant's claim 1. The Examiner suggests that the correct variables appear in blocks of the historical error logs. CUDDIHY's Table 1 (col. 6, lines 23-33) merely lists a set of blocks and shows the weight assigned to each block as being an inverse function of the number of occurrences of the block in an error log. The recited "correct variable" is a program variable that has a correct value during program execution. The blocks of CUDDIHY's Table 1 refer to sections of an error log as illustrated for example in CUDDIHY's FIG. 2. Nothing in the error log indicates anything about variables having correct values. Error logs do not keep track of things that are correct. They keep track of events that indicate device failures. Thus the Examiner's assertion that blocks of a historical error log represent the applicant's "correct variables" is incorrect.

The Examiner correctly points out that CLARK and SHIMOMURA do not teach applying a scaling function to the priority value associated with each program statement in the error set as recited in claim 5. CUDDIHY also fails to teach this. The Examiner points out that CLARK and SHIMOMURA mention "prior experience" and "additional heuristics" but the Examiner does not indicate how such references in CLARK and SHIMOMURA teach the specific limitations of claim 5 relating to the use of scaling functions derived to reduce computed probabilities that a program statement included in both an error set and a sensitized set is an error source. CLARK, SHIMUMURA and CUDDIHY all fail to teach error sets, sensitized sets, or reducing a computed probabilities that a particular program statement is the source of an error in a variable value by a scaling factor determined on the basis of whether the program statement produced correct variable values in the past. Claim 5 is therefore patentable over the cited references.

Claim 6

Claim 6 depends on claim 5 and is patentable over the cited references for similar reasons. Claim 6 further recites that "the scaling function adds a constant value to the priority value …" The Examiner cites col. 5, lines 31-34 as teaching this, but does not indicate which of the three cited references contains these lines. The applicant has reviewed col. 5, lines 31-34 of each of CUDDIHY, CLARK and SHIMIMURA and has found no references to adding a constant to any kind of scaling factor for scaling a probability that a particular program statement in an error set is the source of an error in the value of a program variable. CUDDIHY's col. 5, lines 31-34 talk about how weights of blocks of an error log are established, but this has nothing to do with scaling a probability that a particular program statement in and error set is the source of and error in the value of a program variable.

Claim 7

Claim 7 depends on claim 5 and is patentable over the cited references for similar reasons. Claim 7 further recites that "a second sensitized set comprising at least a program codes statement that is rationally connected to the error variable and for each program code statement in the second sensitized set, the scaling function is applied to the priority value associated with the identical program statement in the error set. The Examiner apparently cites Table 1 of CUDDIHY as teaching this, but CUDDIHY's Table 1 (col. 6, lines 23-33) merely lists a set of blocks and shows a weight assigned to each block as being an inverse function of the number of occurrences of the block in an error log. The weights indicate an error block's value in identifying the source of an equipment error. The weights do not relate to a scaling factor for a probability that a particular statement in a program caused an error in the value of a particular program variables and the weights are determined the basis of a number of occurrences of blocks in an error log and not on whether a program statement occurs in both an "error set" and a "second sensitized set" of program statements as recited in Claim 7.

The cited references therefore fail to teach the additional limitations of claim 7.

Claim 8

Claim 8 depends on claim 5 and is patentable over the cited references for similar reasons. Claim 8 further recites "a plurality of execution cycles prior to the error cycle are utilized to assign the priority values." The Examiner cites CLARK, col. 6, lines 8-12 as teaching weights of program code statements are revised as a result of prior experience or a change in heuristic. However CLARK's weights related to the complexity of a program statement, not a priority value indicating a probability that the statement caused a particular program variable to take on an undesired value at some point during program execution. The fact that it is known to revise a weight characterizing the complexity of a statement based on "prior experiences" or "heuristics" that have nothing to do with execution cycles of a program does not render it obvious to revise a completely unrelated priority value based on information obtain during a plurality of execution cycles of a program.

Claim 9

Claim 9 depends on claim 5 and is patentable over the cited references for similar reasons. Claim 9 further recites "a plurality of correct variables are utilized to assign the priority values." The Examiner cites CLARK, FIG. 3, item 52 as teaching this. Item 52 says "assign node value to each basic block" and CLARK col. 4, lines 38-49 indicates that a "basic block" is a straight line code run (i.e., a sequence of program statements that does not include a conditional statement). CLARK does not clearly indicate what the "value" assigned to each block is, but it appears to be simply a unique number identifying that particular sequence of statements as a node in a graph per col. 4, lines 57-60. Thus CLARK's FIG. 3 item 52 apparently just indicates that various sections of code should be uniquely numbered. This has nothing to do with the additional subject matter of the applicant's claim 9.

Claim 10

Claim 10 depends on claim 1 and is patentable over the cited references for similar reasons.

Claim 15

Claim 15 depends on claim 12 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 5.

Claim 16

Claim 16 depends on claim 15 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 6.

Claim 17

Claim 17 depends on claim 15 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 7.

Claim 18

Claim 18 depends on claim 15 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 8.

Claim 19

Claim 19 depends on claim 15 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 9.

Claim 20

Claim 20 depends on claim 11 and is patentable over the cited references for similar reasons.

Claim 25

Claim 25 depends on claim 24 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 5.

Claim 26

Claim 26 depends on claim 25 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 6.

Claim 27

Claim 27 depends on claim 25 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 7.

Claim 28

Claim 28 depends on claim 25 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 8.

Claim 29

Claim 29 depends on claim 25 and is patentable over the cited references for similar reasons and for reasons similar to those discussed above in connection with claim 9.

Claim 30

Claim 30 depends on claim 21 and is patentable over the cited references for similar reasons.

In view of the foregoing amendment and remarks it is believed the application is in condition for allowance. Notice of Allowance is therefore respectfully requested.

Respectfully submitted,

Daniel J Bedell

Daniel J. Bedell
Reg. No. 30,156

SMITH-HILL & BEDELL, P.C.
12670 NW Barnes Road, Suite 104
Portland, Oregon 97229

Tel. (503) 574-3100
Fax (503) 574-3197
Docket: SPRI 3135

Certificate of Facsimile Transmission

I hereby certify that this paper is being facsimile transmitted to the Patent and Trademark Office on the date shown below.

Penelope Stockwell

Penelope Stockwell

Sept. 8, 2004

Date